REMARKS

Claims 2-8, 10-14, 17 and 19 remain in the application. Applicant asserts that no new matter has been added. Reconsideration of the Application is hereby requested.

Claim Rejections

Rejections Under 35 U.S.C. § 102

Claim 14 and 19 were rejected under 35 U.S.C. § 102(b), as being anticipated by Chiarot et al. (US 5,721,864). Applicant respectfully traverses this rejection for the reasons that follow.

Claim 19 recites the limitations of loading a speculative load into a pipeline and loading a non-speculative load into the pipeline a predetermined number of cycles thereafter. However, Chiarot et al. does not disclose the loading of any type of load into a pipeline. In computer architecture, the term "pipeline" has long been known to mean a set of data processing elements connected in series so that the output of one element is the input of the next one. The elements of a pipeline are executed simultaneously so that the average time to execute each instruction is reduced. This definition is supported by the following exhibits attached hereto: Exhibit A, the Wikipedia entry discussing computer pipelining, accessible on the Internet at http://en.wikipedia.org/wiki/Pipelining; and Exhibit B, a tutorial on computer pipelining by Prof. Gurpur M. Prabhu of the Computer Science department of Iowa State University, accessible on the Internet at http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html. This definition has also long been recognized by the U.S. Patent Office (as evidenced, e.g., by the following U.S. Patent Nos: 3,840,861, issued to Amdahl; 3,949,379, issued to Ball; and 5,440,717, issued to Bosshart).

Nowhere does Chiarot disclose loading anything into a pipeline. The only instance in which Chiarot even uses the word "pipeline" is found in col. 1, line 35, in which he states (as part of the Background section) that pipelines "continue to improve the performance of processing systems." This statement does not relate in any way to the system disclosed in Chiarot and nothing in Chiarot mentions any sort of loading anything into a pipeline. Nowhere else is there an explicit mention of a pipeline, nor is there any disclosure of any type of architecture that includes a series of processing elements that execute more than one instruction simultaneously. Therefore, Chiarot simply fails to disclose a pipeline and, therefore, completely fails to disclose the limitations of "loading a speculative load into the pipeline" and "loading a non-speculative load into the pipeline," as recited in Claim 19. Because Chiarot completely fails to disclose a pipeline and completely fails to disclose loading anything into a pipeline, it cannot anticipate Claim 19, which specifically recites limitations regarding operations performed on a pipeline. This reason is sufficient, by itself, to overcome this rejection.

Chiarot simply discloses a system in which "instructions are pre-fetched into an L1 cache only from an L2 cache, and not from main memory, prior to the resolution of any unresolved branches in the pending instructions." (Chiarot, col. 2, ll. 56-61) While the entry in the L1 cache could be speculative (Chiarot, Abstract), there is no disclosure at all of any sort of non-speculative load being loaded into anything "a predetermined number of cycles after … loading a speculative load," as recited in Claim 19. This reason is also sufficient, by itself, to overcome this rejection.

Claim 14 recites the step of "flushing [a] flagged dependent instruction from the pipeline upon the determination of a misprediction of a corresponding data load." The Action asserts that this limitation is disclosed in Chiarot, citing col. 4, ll. 66-67 and FIG. 2. However, nowhere does Chiarot disclose that *any* instruction is flagged and the cited portion states only that if an L1 cache request for an instruction line will result in a cache miss, the flow diagram will be re-entered at step 205 (entry into the routine shown in FIG. 2), preventing the prefetching of instructions that may be cancelled. This merely states that if there is a cache miss, then the system will re-fetch the current instruction, thereby preventing the fetching of dependent instructions. There is nothing in this section that even suggests that dependent instructions even exist in a pipeline, much less are flushed if there is a misprediction. This reason is sufficient, by itself, to overcome this rejection.

For these reasons, each of which is sufficient by itself to overcome this rejection, Applicant believes that this rejection has been overcome and respectfully requests that it be withdrawn.

Rejections Under 35 U.S.C. § 103

Claims 2-6, 10-13 and 17 were rejected under 35 U.S.C. § 103(a), as being unpatentable over Chiarot in view of Au (US 5,548,795).

The Action, on page 5, admits that "Chiarot et al. do not disclose expressly that the computer architecture is implemented with 'a pipeline'." However, it then states the "Chiarot et al. disclose the 'deeper piplines' … in the admitted prior arts section." (Both references to Chiarot being to U.S. 5,721,864 – the same patent number!) It appears as though the Action is attempting to assert that, while the Chiarot reference fails to disclosed the limitations of a claim, that these limitations are obvious over Chiarot in view of Chiarot. Applicant respectfully asserts that a claim cannot be obvious over a reference in view of itself.

Applicant reasserts the arguments made with respect to the §102 rejections to support its assertion that Chiarot fails to disclose, or otherwise teach or suggest, any sort of pipeline. Applicant also asserts that Au also completely fails to disclose or otherwise teach or suggest, any sort of pipeline. Au merely teaches a disk drive command queue. (Au, Abstract) Because neither Chiarot nor Au teach or suggest a pipeline, as recited in Claim 17, Applicant respectfully asserts that this rejection has been overcome.

Furthermore, to render Claim 17 obvious, the cited references would have to show at least the following elements: (1) a pipeline; (2) a fast-load cache that loads a speculative load into the pipeline; and (3) an L1 cache that loads a non-speculative load corresponding to the speculative load into the pipeline "a predetermined number of cycles after the fast-load data cache loads the speculative data load." While the action fails to identify anything that is a pipeline in the cited references, if the caches disclosed in Chiarot are part of a pipeline, as the Action seems to imply, then they could not satisfy the limitations relating to the things that load the speculative load and the non-speculative load into the pipeline.

Also, there is no disclosure in either Chiarot or Au of any fixed delay ("a predetermined number of cycles") between the loading of different instruction lines. It appears as though data is fetched into the L2 cache in Chiarot is only upon a random cache miss event, not after a fixed predetermined number of cycles, as recited in Claim 17.

Neither Chiarot nor Au teach or suggest, either alone or in combination, the loading of a speculative load into a pipeline and then the loading of a non-speculative load into the pipeline a predetermined number of cycles after the loading of the speculative load. For this reason, it is believed that this rejection has been overcome and Applicant respectfully requests that it be withdrawn.

Prior Art Made of Record

In addition to the remarks presented above, Applicant asserts that the remaining prior art made of record neither anticipates, nor renders obvious the claimed invention.

CONCLUSION

Applicant believes that the rejections have been overcome for the reasons recited above. Therefore, Applicant respectfully requests that all remaining claims be allowed and that a timely Notice of Allowance be issued.

No addition fees are believed due. However, the Commissioner is hereby authorized to charge any additional fees that may be required, including any necessary extensions of time, which are hereby requested, to Deposit Account No. 503535.

09/17/2008
Date

[illegible]
Bryan W. Bockhop
Registration No. 39,613

Customer Number: 25854

Bockhop & Associates, LLC
2375 Mossy Branch Dr.
Snellville, GA 30078

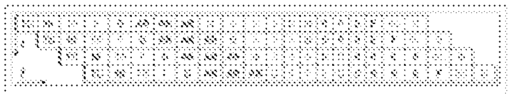
Tel. 678-919-1075
Fax 678-609-1483
E-Mail: bwb@bockpatent.com

EXHIBIT A

Learn more about using Wikipedia for research.

Pipeline (computing)

From Wikipedia, the free encyclopedia
(Redirected from Pipelining)


Instruction scheduling on the Intel Pentium 4.

In computing, a **pipeline** is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements.

Computer-related pipelines include:

- Instruction pipelines, such as the classic RISC pipeline, which are used in processors to allow overlapping execution of multiple instructions with the same circuitry. The circuitry is usually divided up into stages, including instruction decoding, arithmetic, and register fetching stages, wherein each stage processes one instruction at a time.
- Graphics pipelines, found in most graphics cards, which consist of multiple arithmetic units, or complete CPUs, that implement the various stages of common rendering operations (perspective projection, window clipping, color and light calculation, rendering, etc.).
- Software pipelines, consisting of multiple processes arranged so that the output stream of one process is automatically and promptly fed as the input stream of the next one. Unix pipelines are the classical implementation of this concept.

Contents

Concept and motivation

Pipelining is a natural concept in everyday life, e.g. on an assembly line. Consider the assembly of a car: assume that certain steps in the assembly line are to install the engine, install the hood, and install the wheels (in that order, with arbitrary interstitial steps). A car on the assembly line can have only one of the three steps done at once. After the car has its engine installed, it moves on to having its hood installed, leaving the engine installation facilities available for the next car. The first car then moves on to wheel installation, the second car to hood installation, and a third car begins to have its engine installed. If engine installation takes 20 minutes, hood installation takes 5 minutes, and wheel installation takes 10 minutes, then finishing all three cars when only one car can be operated at once would take 105 minutes. On the other hand, using the assembly line, the total time to complete all three is 75 minutes. At this point, additional cars will come off the assembly line at 20 minute increments.

Costs, drawbacks, and benefits

As the assembly line example shows, pipelining doesn't decrease the time for a single datum to be processed; it only increases the throughput of the system when processing a stream of data.

High pipelining leads to increase of latency - the time required for a signal to propagate through a full pipe.

A pipelined system typically requires more resources (circuit elements, processing units, computer memory, etc.) than one that executes one batch at a time, because its stages cannot reuse the resources of a previous stage. Moreover, pipelining may increase the time it takes for an instruction to finish.

Design considerations

One key aspect of pipeline design is balancing pipeline stages. Using the assembly line example, we could have greater time savings if both the engine and wheels took only 15 minutes. Although the system latency would still be 35 minutes, we would be able to output a new car every 15 minutes.

Another design consideration is the provision of adequate buffering between the pipeline stages — especially when the processing times are irregular, or when data items may be created or destroyed along the pipeline.

Implementations

Buffered, Synchronous pipelines

Conventional microprocessors are synchronous circuits that use buffered, synchronous pipelines. In these pipelines, "pipeline registers" are inserted in-between pipeline stages, and are clocked synchronously. The time between each clock signal is set to be greater than the longest delay between pipeline stages, so that when the registers are clocked, the data that is written to them is the final result of the previous stage.

Buffered, Asynchronous pipelines

Asynchronous pipelines are used in asynchronous circuits, and have their pipeline registers clocked asynchronously. Generally speaking, they use a request/acknowledge system, wherein each stage can detect when it's "finished". When a stage is finished and the next stage has sent it a "request" signal, the stage sends an "acknowledge" signal to the next stage, and a "request" signal to the previous stage. When a stage receives an "acknowledge" signal, it clocks its input registers, thus reading in the data from the previous stage.

The AMULET microprocessor is an example of a microprocessor that uses buffered, asynchronous pipelines.

Unbuffered pipelines

Unbuffered pipelines, called "wave pipelines", do not have registers in-between pipeline stages. Instead, the delays in the pipeline are "balanced" so that, for each stage, the difference between the first stabilized output data and the last is minimized. Thus, data flows in "waves" through the pipeline, and each wave is kept as short (synchronous) as possible.

The maximum rate that data can be fed into a wave pipeline is determined by the maximum difference in delay between the first piece of data coming out of the pipe and the last piece of data, for any given wave. If data is fed in faster than this, it is possible for waves of data to interfere with each other.

References

- For a standard discussion on pipelining in parallel computing see "Parallel Programming in C with MPI and OpenMP" by Michael J. Quinn,McGraw-Hill Professional, 2004

See also

- Throughput
- Parallelism
- Instruction pipeline
  - Classic RISC pipeline
- Graphics pipeline
- Pipeline (software)
  - Pipeline (Unix)
  - Hartmann pipeline for VM
  - BatchPipes for MVS
- Geometry pipelines
- XML pipeline

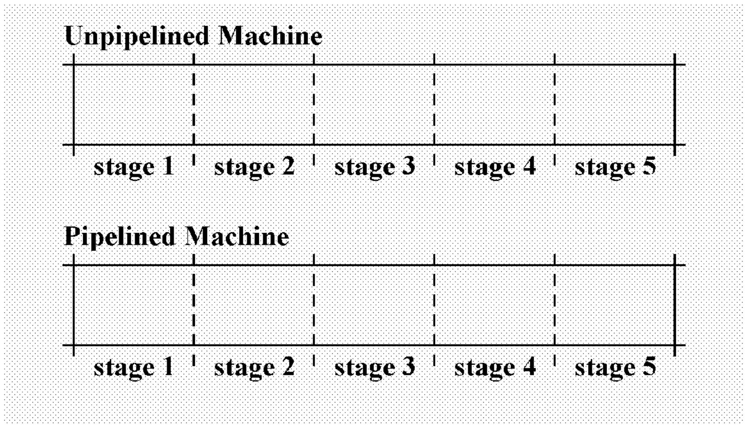
Retrieved from "http://en.wikipedia.org/wiki/Pipeline_(computing)"
Categories: Instruction processing

by: Gurpur M. Prabhu
Associate Professor of Computer Science
Iowa State University

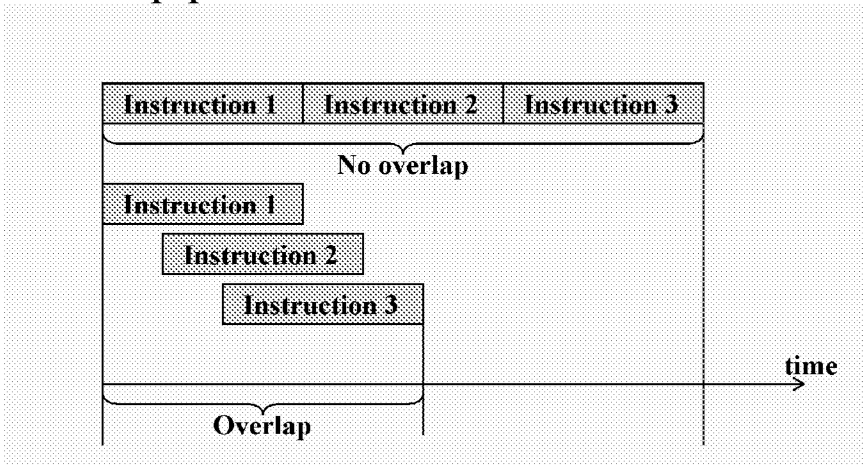
EXHIBIT B

PIPELINING

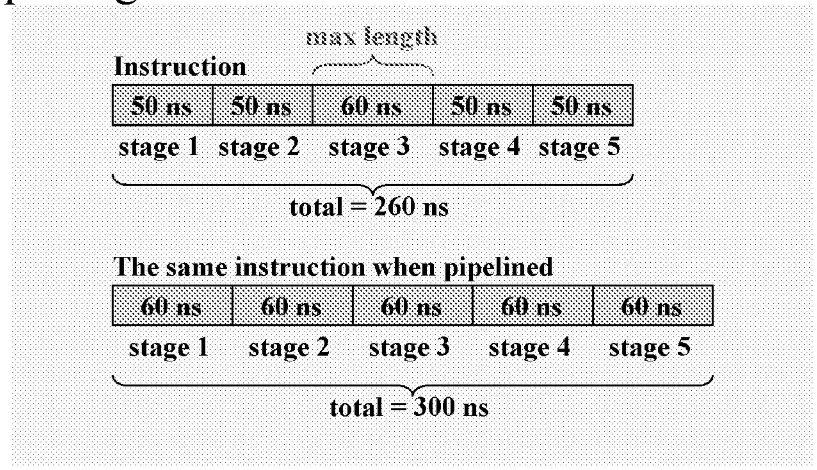
Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in **stages**. Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.



Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The **throughput** of the instruction pipeline is determined by how often an instruction exits the pipeline.



Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time. We call the time required to move an instruction one step further in the pipeline ***a machine cycle*** . The ***length*** of the machine cycle is determined by the time required for the slowest pipe stage.



The pipeline designer's *goal is to balance the length of each pipeline stage* . If the stages are perfectly balanced, then the time per instruction on the pipelined machine is equal to

$$\frac{\text{Time per instruction on nonpipelined machine}}{\text{Number of pipe stages}}$$

Under these conditions, the speedup from pipelining equals the number of pipe stages. Usually, however, the stages will not be perfectly balanced; besides, the pipelining itself involves some overhead.

We will describe the principles of pipelining using DLX and a simple version of its pipeline. Those principles apply to more complex instruction sets than DLX , although the resulting pipelines are more complex.